







# Javascript



DESIGN IS NOT JUST WHAT IT LOOKS LIKE  
AND FEELS LIKE  
DESIGN IS HOW IT WORKS

STEVE JOBS

NFM  
*nunoferreira.com*

*Design não é só como uma coisa aparenta ou como se sente, design é como ela funciona*

# #Introdução Javascript

JS



- linguagem de programação interpretada
- Originalmente implementada para navegadores web (scripts lado cliente)
- Controlando o navegador com comunicação assíncrona
- Linguagem mais popular do mundo
- Principal linguagem de programação client-side em navegadores web
- utilizada do lado do servidor através de ambientes como o **node.js**
- Orientação a objetos baseada em protótipos
- Tipagem fraca
- Suporte a programação funcional - funções de primeira classe
- apresenta recursos como fechamentos
- funções de alta ordem comumente (não disponível em Java ou C++)
- Baseado no ECMAScript (Ecma international nas especificações ECMA-2623 e ISO/IEC 16262)



# #Introdução Javascript

JS



Web Notícias Imagens Vídeos Shopping Mais Ferramentas de pesquisa



Aproximadamente 365.000 resultados (0,29 segundos)

Dica: Pesquisar apenas resultados em português (Brasil). Especifique seu idioma de pesquisa em Preferências

## Brendan Eich – Wikipédia, a enciclopédia livre

[pt.wikipedia.org/wiki/Brendan\\_Eich](https://pt.wikipedia.org/wiki/Brendan_Eich)

Brendan Eich conseguiu seu bacharelado em matemática e ciência da computação ... Eich doou US\$1,000 em 2008 para a campanha de suporte à Califórnia ...

## Brendan Eich - Wikipedia, the free encyclopedia

[en.wikipedia.org/wiki/Brendan\\_Eich](https://en.wikipedia.org/wiki/Brendan_Eich) Traduzir esta página

Brendan Eich is an American technologist and creator of the JavaScript programming language. He co-founded the Mozilla project, the Mozilla Foundation and ...

## Brendan Eich

<https://brendaneich.com/> Traduzir esta página

3 de abr de 2014 - Inventor of JavaScript. Occasional talks about the future of the language with references to its history.

## BrendanEich (@BrendanEich) | Twitter

<https://twitter.com/brendaneich> Traduzir esta página

26.3K tweets • 153 photos/videos • 38.4K followers. "@0xabad1dea http://t.co/trvK0N1vya "Look around you, can you form some sort of rudimentary lathe?"

## Backlash Against Brendan Eich Crossed A Line - Forbes

[www.forbes.com/.../backlash-against-brendan-eich-c...](http://www.forbes.com/.../backlash-against-brendan-eich-c...) Traduzir esta página

5 de abr de 2014 - Brendan Eich recently stepped down as CEO of Mozilla, developer of the Firefox Web browser. It may be more accurate to say he was forced ...



Mais imagens

## Brendan Eich

Programador

Brendan Eich é um programador de computadores americano e criador da linguagem de programação JavaScript. Ex-chefe do escritório de tecnologia na Mozilla Corporation. [Wikipédia](#)

**Nascimento:** 1961, Pittsburgh, Pensilvânia, EUA

**Educação:** Universidade de Illinois em Urbana-Champaign, Universidade de Santa Clara

## Pesquisas relacionadas

Ver mais 15



Mitchell Baker



Douglas Crockford



Chris Beard



John Resig



Paul Irish

# #Evolução Javascript

JS



2014

2008

2005

2000

1998

1997

09/1996

03/1996

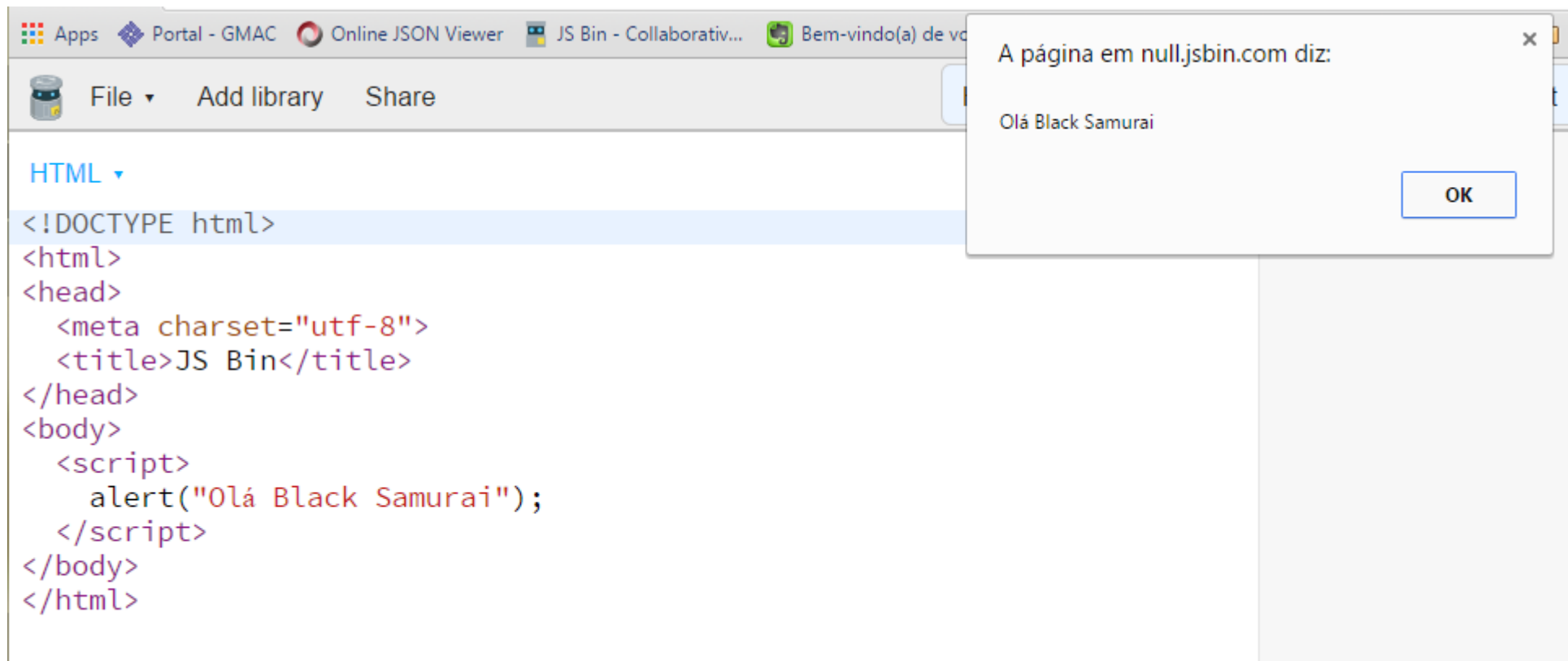


# #Testando Javascript

JS



- A execução do código é instantânea



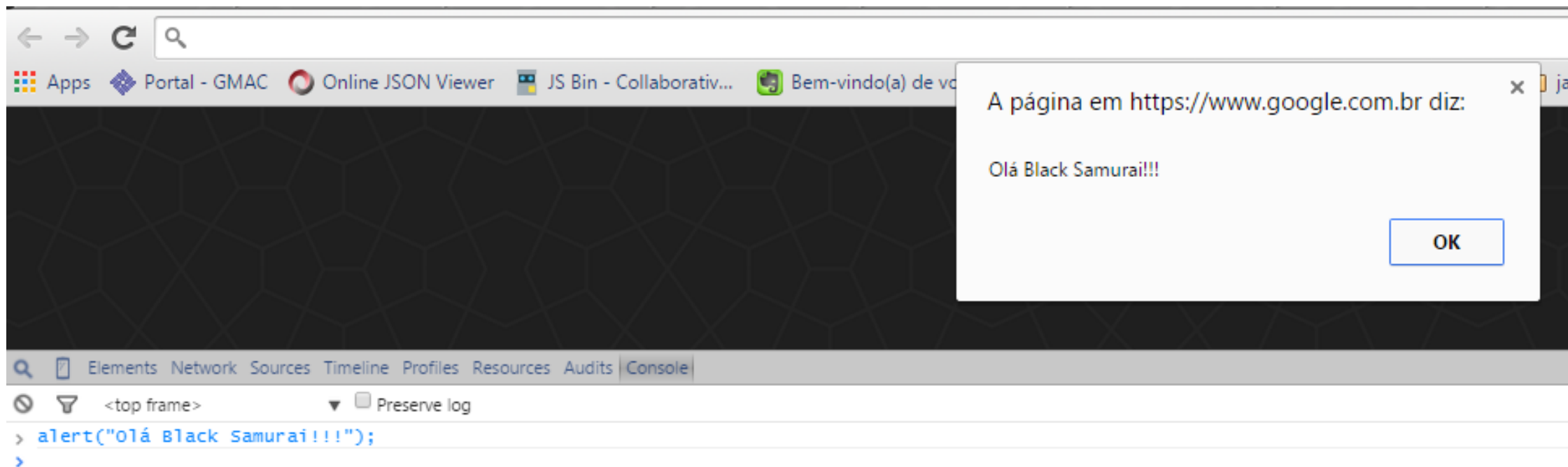


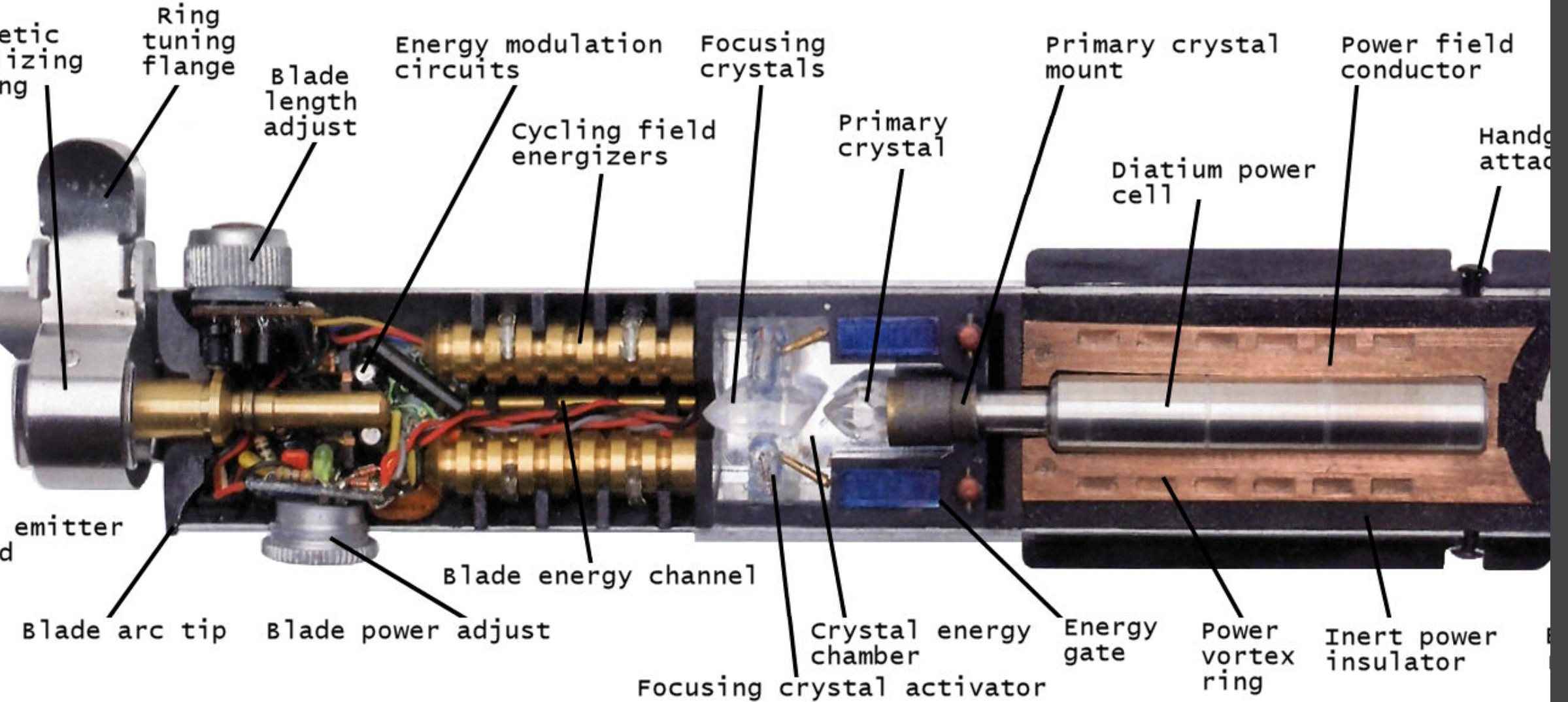
# #Testando Javascript

JS



- Usando o console do navegador





# Sintaxe Javascript

# #Operações Matemáticas

JS



- Soma  $+$
- Subtração  $-$
- Divisão  $/$
- Multiplicação  $*$
- Resto da Divisão  $\%$
- $++$ ,  $--$ ,  $+=$ ,  $-=$ ,  $*=$ ,  $/=$ ,  $\%=$

# #Operações Matemáticas

JS



- Teste no console:

```
> 10+13
< 23
> 14*3
< 42
> 20-4
< 16
> 25/5
< 5
> 23%2
< 1
> |
```



# #Operações Matemáticas

JS



- Incremento e decremente

```
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame> Preserve log
> var a = 2;
< undefined
> console.log('incremento', a++);
incremento 2
< undefined
> a
< 3
> console.log('decremento', a--);
decremento 3
< undefined
> a
< 2
> |
```

# #Funções Matemáticas

JS



Função	Exemplo	Comentário
abs	<code>y = Math.abs(x);</code>	Retorna o valor absoluto de um número (x no exemplo).
ceil	<code>i = Math.ceil(x);</code>	Retorna o menor inteiro maior ou igual ao número dado. Por exemplo: para $x = 30.75$ retorna 31 e para $x = -30.75$ retorna -30.
cos	<code>y = Math.cos(x);</code>	Retorna o co-seno de um número que representa um ângulo em radianos (x no exemplo). O resultado, conforme definição matemática da função, está na faixa de -1 a 1.
exp	<code>y = Math.exp(x);</code>	Retorna o número e (base dos logaritmos naturais) elevado ao argumento (x no exemplo).
floor	<code>i = Math.floor(x);</code>	Retorna o maior inteiro menor ou igual ao número dado. Por exemplo: para $x = 30.75$ retorna 30 e para $x = -30.75$ retorna -31.
log	<code>y = Math.log(x);</code>	Retorna o logaritmo natural (base e) de um número (x no exemplo). Se $x = 0$ , retorna -Infinity. Se $x < 0$ , retorna NaN porque está fora da faixa.
pow	<code>p = Math.pow(x,y);</code>	Retorna a base elevada ao expoente. No exemplo dado abaixo, x é a base e y é o expoente, isto é, $p = xy$ .
sqrt	<code>y = Math.sin(x);</code>	Retorna o seno de um número que representa um ângulo em radianos (x no exemplo). O resultado, conforme definição matemática da função, está na faixa de -1 a 1.

# #Operações Condicionais

JS



- > Maior
- >= Maior ou igual
- < Menor
- <= Menor ou igual
- == Igual
- != Diferente
- === Igual
- !== Diferente

# #Operações Condicionais

JS



- Testando no console

```
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame> Preserve log
> //valores iguais
15 == 15;
< true
> //valores e tipos iguais
15 === '15';
< false
> //valores diferente
15 != '15';
< false
> //valores e tipos diferentes
15 !== '15';
< true
> //maior que
15 > 10;
< true
> //maior que
15 >= 10;
< true
> //menor que
15 < 10;
< false
> //menor igual que
15 <= 16;
< true
> |
```



# #Operadores logicos

JS



- **&&** e lógico (AND)
- **||** ou lógico (OR)
- **!** não lógico (not)

Tabela E	Tabela OU	Tabela NÃO
$V \text{ e } V \rightarrow V$	$V \text{ ou } V \rightarrow V$	$\text{N\~ao } V \rightarrow F$
$V \text{ e } F \rightarrow F$	$V \text{ ou } F \rightarrow V$	$\text{N\~ao } V \rightarrow F$
$F \text{ e } V \rightarrow F$	$F \text{ ou } V \rightarrow V$	
$F \text{ e } F \rightarrow F$	$F \text{ ou } F \rightarrow F$	

# #Operadores logicos

JS



```
Elements Network Sources Timeline Profiles Resources Audits Console
< top frame > [x] Preserve log
> 15 == 15 && 'Elisio' === 'Elisio'
< true
> 15 == 15 && 'Elisio' === 'carlos'
< false
> //&& = and => retorna verdadeiro se todos forem verdadeiros
< undefined
> 15 == 15 || 'Elisio' === 'carlos'
< true
> 15 == 10 || 'Elisio' === 'carlos'
< false
> // || = or => retorna verdadeiro se um for verdadeiro
< undefined
> !(15 == 15)
< false
> !(15 == 10)
< true
> // ! = negação => nega a saída da operação
< undefined
> |
```

# #Operadores de atribuição

JS



Elements Network Sources Timeline Profiles Resources Audits Console

<top frame> Preserve log

```
> var black = 'samurai';  
// black '=' recebe samurai  
< undefined  
> var op2 = 2;  
var op4 = 4;  
op4 += op2;  
console.log(op4);  
6  
< undefined  
> // com tipos inteiros '+' soma e atribui  
< undefined  
> var op2 = 'Black';  
var op4 = 'Samurai';  
op4 += op2;  
console.log(op4);  
SamuraiBlack  
< undefined  
> // com tipos string '+' concatena as strings  
< undefined  
> 'Black' + 'Samurai';  
< "Black Samurai"  
> |
```

# #Operadores bit a bit

JS



NOT bit a bit

~

Deslocamento para a esquerda bit a bit

<<

Deslocamento para a direita bit a bit

>>

Deslocamento para a direita sem sinal

>>>

AND bit a bit

&

XOR bit a bit

^

OR bit a bit

|



# #Operadores diversos

JS



---

delete

delete

---

typeof

typeof

---

void

void

---

instanceof

instanceof

---

new

new

---

in

in

---

# #Comentarios

JS



- De linha
  - `//Comentário`
- De bloco
  - `/* Comentário de uma ou mais linhas */`



# #Estrutura condicional

```
//Condição tradicional
if (15>10) {
    document.write('Maior');
} else if(15==10) {
    document.write('Igual');
} else{
    document.write('Menor');
};

//Condição Resumida
var maior = (15>10)? 'maior' : 'menor';

//Switch - case
switch(variavel) {
    case 1:
        document.write('Opção 1');
        break;
    case 2:
        document.write('Opção 2');
        break;
    case 3:
        document.write('Opção 3');
        break;
    default:
        document.write('Padrão');
        break;
}
```



# #Estrutura repetição

// --- FOR

```
for (i=0; i<= 10; i++) {  
    document.write('Linha '+i);  
}
```

// --- WHILE

```
var var1 = 0;  
while (var1 <= 10) {  
    document.write('linha '+var1);  
    var1++;  
}
```



# #Estrutura Exceção

- **Throw** - Gera uma condição de erro que pode ser manipulada pelo try/catch/finally

```
var codigo = 8;

try {
    if (codigo < 10) {
        throw "erro_1";
    } else if(codigo > 15){
        throw "erro_2";
    }
} catch (erro) {
    if (erro == "erro_1") {
        alert('Codigo é menor que 10');
    }
    if (erro == "erro_2") {
        alert('Codigo é maior que 15');
    }
} finally{
    alert('Codigo passa por aqui');
}
```

# #Variavel

JS

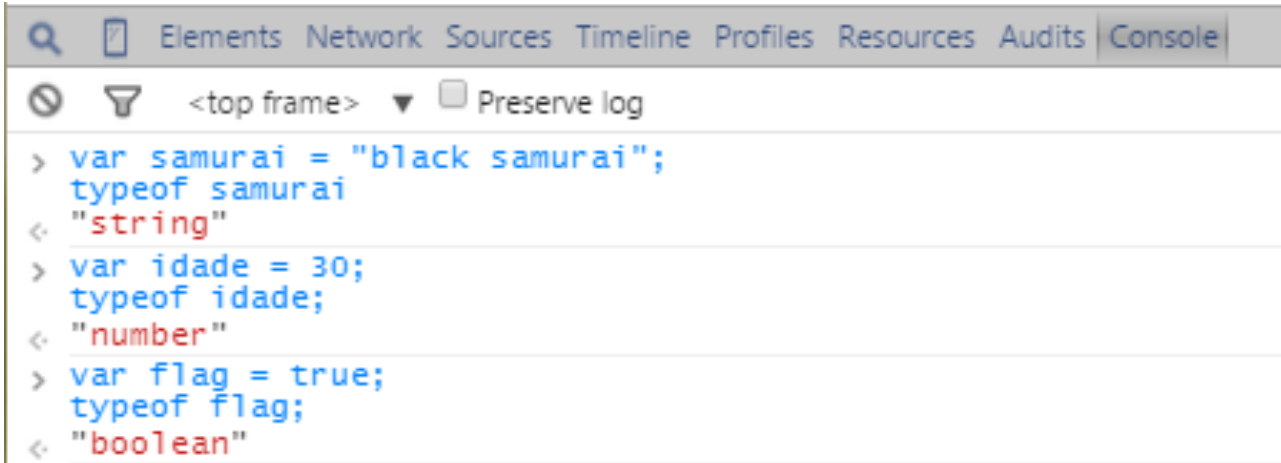


- Armazena um valor para uso posterior:

```
> var contador = 0;  
< undefined  
> contador++;  
< 0  
> contador  
< 1  
> contador++;  
< 1  
> contador;  
< 2  
> |
```

# #Tipos de dados

- Tipos primitivos (armazenados em variaveis):
- String(letras e palavras),
- Number(inteiros decimais),
- Boolean(verdadeiro ou falso),



```
> var samurai = "black samurai";
    typeof samurai
< "string"
> var idade = 30;
    typeof idade;
< "number"
> var flag = true;
    typeof flag;
< "boolean"
```

# #Tipos de dados

- Tipos Especiais
- Null(valor nulo, bom para inicializar uma variavel)
- Undefined(variável não foi assinado ou inicializada)

```
> var nulo = null  
nulo;  
< null  
> var elisio;  
typeof elisio;  
< "undefined"  
> |
```

# #Tipos de dados

JS



- Tipos Objetos
- Object(objeto)
- Array (vetores)
- Funções
- Date (data)
- RegExp (Expressão regular)
- Error

```
Elements Network Sources Timeline Profiles Resources Audits Console
< top frame > Preserve log

> /* Tipos de Objeto */
< undefined
> // objeto do tipo array (vetor)
var guitarras = [ 'ibanez', 'music man', 'suhr' ];
typeof guitarras; // -> "object"
< "object"
> // objeto do tipo function
var soma = function( valor1, valor2 ) { return valor1 + valor2; }
typeof soma; // -> "function"
< "function"
> // objeto do tipo Date
var agora = new Date();
typeof agora; // -> "object"
< "object"
> // objeto do tipo RegExp
var minhaRegExp = /padrao/;
typeof minhaRegExp; // -> "object"
< "object"
> // objeto do tipo Error
var perigo = new Error( 'Alguma coisa deu errado!' );
typeof perigo; // -> "object"
< "object"
> var Jedi = new Object();
typeof Jedi;
< "object"
```

# #Palavras reservadas

JS



abstract  
boolean  
break  
byte  
case  
catch  
char  
class  
const  
continue  
debugger  
default  
delete  
do  
double  
else  
enum  
export

extends  
false  
final  
finally  
float  
for  
function  
goto  
if  
implements  
import  
in  
instanceof  
int  
interface  
long  
native  
new

null  
package  
private  
protected  
public  
return  
short  
static  
super  
switch  
synchronized  
this  
throw  
throws  
transient  
true  
try  
typeof

var  
volatile  
void  
while  
with



- Função é um poderoso objeto destinado a executar uma ação
- É um bloco de código capaz de realizar ações
- Função é um exemplo de reutilização inteligente de código
- Tem a finalidade de dar maior legibilidade ao programa e facilitar a manutenção
- Podemos retornar valor através da instrução **return**



# #Funções

JS



- Criando Funções

- Com o uso da declaração function:

```
function minhaFuncao(){  
    // aqui bloco de código  
};
```

- Com o uso da sintaxe literal:

```
var minhaFuncao = function(){  
    // aqui bloco de código  
};
```



# #Atividade:

JS



1. Escreva uma função que mostre os números ímpares entre 1 e 10.
2. Escreva uma função que receba como parâmetro um número de 1 - 7 e retorne o dia da semana, sendo que 1 é segunda. Também faça a validação para um número inválido.
3. Crie uma variável “somar” que receba uma função que some dois números. Em seguida crie uma função “assert” para testar e validar se a função “somar” está retornando corretamente. O assert deve receber dois parâmetros: o resultado e a função que será testada.

# #Resposta atividade 1

JS



1. Escreva uma função que mostre os números ímpares entre 1 e 10.

```
1 //exercício 01
2 //Escreva um código que mostre os números ímpares entre 1 e 10.
3 function impar(){
4     for(var i = 0; i < 10; i++){
5         if(i%2!==0){
6             console.log(i);
7         }
8     }
9 }
10
11 impar();
12
```

# #Resposta atividade 2

2. Escreva uma função no qual passado o numero de 1 - 7 retorne o dia da semana, sendo que 1 é segunda. Também faça a validação para um numero invalido.

```
function diaSemana(dia){
    var stringDay = '';
    switch(dia){
        case 1:
            stringDay = 'Segunda';
            break;
        case 2:
            stringDay = 'Terça';
            break;
        case 3:
            stringDay = 'Quarta';
            break;
        case 4:
            stringDay = 'Quinta';
            break;
        case 5:
            stringDay = 'Sexta';
            break;
        case 6:
            stringDay = 'Sabado';
            break;
        case 7:
            stringDay = 'Domingo';
            break;
        default:
            stringDay = 'Dia invalido';
            break;
    }
    return stringDay;
}

console.log(diaSemana(5));
```

## #Resposta atividade 3

JS



3. Crie uma variável “somar” que receba uma função que some dois números. Em seguida crie uma função “assert” para testar e validar se a função “somar” está retornando corretamente. O assert deve receber dois parâmetros: o resultado e a função que será testada.

```
var somar = function(a,b){  
    return a+b;  
};  
  
function assert(resultado, fn){  
    (resultado === fn)? console.log('Teste Passou'): console.log('Teste Falhou');  
}  
  
assert(4, somar(2,2));
```